

Authentic SPI – SPI/Microwire Virtual Peripherals for SX-28

Synchronous serial interfaces are widely used to provide economical board-level interface between different devices such as microcontrollers, DACs, ADCs and other. Although there is no single standard for the synchronous serial bus, there are industry accepted guidelines based on two most popular implementations: SPI (a trademark of Motorola Semiconductor) and Microwire (a trademark of National Semiconductor). These interfaces use master-slave model and typically have three signal lines: data input line, data output line and clock line. Chip select signals from the master are used to address different slaves on the bus. The hardware realization of such an interface is a simple shift register. The data bits are shifted in/out MSB (most significant bit) first. Often the data is shifted simultaneously out from the output pin and in to the input pin.

The primary difference between the SPI and the Microwire is that SPI supports data locking on both falling and rising edges of the clock signal while the latter is always operating on the rising edge. Many IC manufacturers produce components that are compatible with SPI and Microwire. Since both standards define only the communication lines and the clock edge, other parameters vary for different devices. Clock frequencies happen to be anywhere from 100kHz to a few MHz and word lengths are from 8 to 16 or more bits. This makes it hard to create a single virtual peripheral (VP) that would work for all of these situations. Therefore a family of SPI/Microwire VPs optimized for different operating conditions was created:

SPIM	high bit-rate (1.72 - 0.2 MHz clock frequency) master mode VP
SPIS	high bit-rate (1.1 and lower clock frequency) slave mode VP
SPIML	low bit-rate (330 – 0.7659KHz clock frequency) master mode VP
SPIG	general SPI driver, which supports all the modes

All virtual peripherals support word lengths anywhere from 1 to 16 bits. SPIM VP does not use any interrupts; SPIS uses wake-up interrupts from port B (therefore is limited to PORTB pins only); and the SPI_ML uses RTCC interrupts to generate slow clock while freeing the processor for other activities in between.

Three demo programs were created to demonstrate the operation of SPI virtual peripherals.

The SPI Demo #1 uses SPIML VP working with two serial devices: the 12-bit ADC (Linear Technology LTC1594) and the 12-bit DAC (LTC1453). Special SPI evaluation board was developed for this demo program. The program executes the cycles of analog to digital conversion followed by digital to analog conversion.

The SPI Demo #2 program is similar to the #1 with the difference that the processing of sample data is offloaded to another SX-28 microcontroller. After the sample is acquired by the master SX-28, it is then transferred via high speed SPI to the slave SX-28, processed, transferred back and then output to the DAC.

The SPI Demo #3 demonstrates operation of high-speed data transfer between SPIM and SPIS, running on different SX microcontrollers.

Extensive documentation and source files are included.

```

=====
;
; SERIAL PERIPHERAL INTERFACE (SPI) DRIVER
; SPI_ML
; Revision 1.01, 07.05.98
; Developed by Authentic Engineering for Scenix Semiconductor, Inc.
; Authentic Engineering // www.authenticeng.com // ak@silcom.com
;
=====
; MASTER MODE LOW RATE SPI DRIVER. SINGLE MASTER CONFIGURATION
;
; This driver provides SPI interface to realtively slow devices.
; The data are transfered by words. Word length is software configurable
; from 1 to 16 bits.
; This driver is a part of a set of SX SPI drivers. It makes use of general
; set of control and status data, developed for the entire family of drivers
;
; CLOCK RATE
;
; The SPI clock rate is driven by the system RTCC.
; Minimum clock rate is limited by the RTCC prescaler setting which should be
; configured by the user. When the RTCC is configured to the minimum rate and
; prescaler is set to 1:2, the SPI clock rate will be equal to 10.2 microsec.
; With the prescaler setting at 1:256 the minimum clock rate will be 1.3056 m
; Maximum clock rate can be achived at RTCC prescaler setting 1:2
; and RTCC reload register - $80. Desierable configuration of the RTCC presc
; must be determined by the user.
;
; Clock (SCLK) PHASE and POLARITY are software configurable.
;
; There are FOUR basic signal lines associated with the SPI protocol:
; Master-Out-Slave-In MOSI signal line
; Master-In-Slave_Out MISO signal line
; Serial Clock SCLK signal line
; Slave Select SS signal line
; Arbitrary pins of the SX-28 can be used for current driver
;
=====
;
; INITIALIZATION OF SPI DRIVER
;
; USER program must:
; 1. Provide appropriate configuration of the SX-28 ports.
;
; 2. Define next parameters:
;
; SPI_RAM_BANK - to define the origin in SX RAM banks
; SPI_CLOCK_RATE - to define the clock rate. This byte is used by the driver
; as the RTCC reload register value
; SPI_COMMAND - command and configuration byte. This byte defines:
; - SPI Clock polarity
; - Mode (Master or Slave)
; - Speed (Fast or Slow)
; - Direction (Read/Write)
; - Word Lenght (1 to 16).
;
; Driver will set the RTCC reload value to the SPI_CLOCK_RATE. The USER
; program must take care about restoration of the RTCC reload value, after
; the SPI driver will complete its operation.
;

```

```

; Current driver support only Master Mode and Slow Speed data transfer.
;
; The driver supports 5 commands, specified by the 4 MSBs of the SPI_COMMAND
; $0      RESET
; $2      GET WORD in MASTER mode, define FALLING edge
; $6      GET WORD in MASTER mode, define RISING edge
; $A      SEND WORD in MASTER mode, define FALLING edge
; $E      SEND WORD in MASTER mode, define RISING edge
; Generally for the SPI interface there is no difference between GET and SEND,
; however it seems convinient to have different commands, because it allows t
; transmit and receive cycles for specific peripherals. Such an examle is pre
;
; Four LSBs of SPI_COMMAND byte are used for word length definition.
;
; $1      corresponds to one bit length
; ...
; $F      corresponds to fifteen bit legth
; $0      corresponds to sixteen bit length
;
; For example, SPI_COMMAND for 12 bit data transfer to the peripheral looks l
;
; 3. Driver is driven by RTCC interrupts. In order to start data transfer USE
; must:
; - check that SPI_BUSY flag is cleared
; - set SPI_CLOCK_RATE and SPI_COMMAND
; - enable timer interrupts
; - perform JMP SPI_ENTRY when RTCC interrupt occures
; See example in DEMO #1
;=====
;
; The SPI_STATUS byte provides the information on driver status. It consists
; flags, described in the comments to the code.
;
;
; SX-28 pin assignment presented below is an example used in 'SPI DEMO #1'
; program.
;=====
;                                CODE START
;
;                                device pins28, pages4, banks8
;                                device turbo, stackx, optionx
;                                id      'SPI_ML'
;
;=====
;                                PIN ASSIGNMENT
;=====
;                                example of pin assignment used in DEMO #1
;
spi_clk_pin      =      rb.0      ;SPI clock line
spi_out_pin      =      rb.1      ;master_in slave_out line
spi_in_pin       =      rb.2      ;master_out slave_in line
spi_ADC_cs_pin   =      rb.4      ;slave select (ADC)
spi_DAC_cs_pin   =      rb.5      ;slave select (DAC)
;
;=====
;                                SPI DEVICE RAM ORIGIN
;
spi_ram_bank      org      10h      ;example of RAM bank assignment
                  =      $
spi_command       ds      1
bit_wide_0        equ      spi_command.0      ;the word length LSB
bit_wide_1        equ      spi_command.1      ;1_st bit

```

```

bit_wide_2          equ      spi_command.2      ;2_nd bit
bit_wide_3          equ      spi_command.3      ;the word length MSB
fast_ON             equ      spi_command.4      ;set if FAST MODE
master_ON           equ      spi_command.5      ;set if MASTER MODE
rising_ON           equ      spi_command.6      ;set if RISING EDGE
transmit_ON         equ      spi_command.7      ;set if TRANSMIT

;=====
spi_status           ds      1                  SPI_STATUS byte
spi_ready            equ      spi_status.0      ;SET when driver is ready (has been

spi_busy             equ      spi_status.1      ;set when data transfer is active
;spi_master          equ      spi_status.2      ;SET when SPI master is initied
;spi_slave           equ      spi_status.3      ;SET when SPI slave is initied

spi_command_ERR      equ      spi_status.4      ;set when command byte contains no
spi_tx_complete      equ      spi_status.5      ;transmit complete
spi_rx_complete      equ      spi_status.6      ;receive complete
;spi_error           equ      spi_status.7      ;SET when transmit is not finished

;spi_rx_complete and spi_tx_complete flags are needed when two different SS lines w
;and receive and transmit are controled in the same time. Otherwise, this flag is n
;(see SPI_M, SPI_S SPI drivers).

;=====
spi_state            ds      1                  SPI_STATE byte

spi_ms_ON            equ      spi_state.0      ;set when Master Mode
spi_sl_ON            equ      spi_state.1      ;set when Slave Mode
spi_rx_ON            equ      spi_state.2      ;set when Receive Mode
spi_tx_ON            equ      spi_state.3      ;set when Transmit Mode
spi_clk_state        equ      spi_state.4      ;state of the SPI_CLOCK line;
;spi_timer_IRQ_ON    equ      spi_state.5      ;SPI transfer under timer IRQ      :in
;spi_port_IRQ_ON     equ      spi_state.6      ;SPI transfer under port IRQ       :in

;=====
spi_clk_rate         ds      1                  SPI_CLOCK byte
;RTCC reload value

spi_buff             ds      1                  ;SPI I/O shift register (MSByte)
spi_buff_low         ds      1                  ;SPI I/O shift register (LSByte)
spi_shift_counter    ds      1                  ;shift counter

spi_bits_total       ds      1                  ;number of bits to transfer
spi_shifts_init      ds      1                  ;number of initial shifts before tr

spi_clk_counter      ds      1                  ;used only for DEMO #1 with LTC1594
spi_clks_init        ds      1                  ;used only for DEMO #1 with LTC1594

;=====
INTERRUPT START

spi_entry            bank    spi_ram_bank      ;SPI driver entry point
                    jb       master_ON, master_entry

                    snb       rising_ON
                    setb      spi_command_ERR    ;return from interrupt if wrong com
                    snb       transmit_ON
                    setb      spi_command_ERR

                    clr       spi_command

```

```

        setb    spi_ready

        mov     w,spi_clk_rate
        retiw

;=====
master_entry    test    spi_shift_counter    ;test if this is first entr
                jnz     :master_start

:init_master    jlb     spi_ms_ON,:ms_rx_finish ;initialization of the driv

                setb    spi_busy_ON
                setb    spi_ms_ON

                sb      transmit_ON
                setb    spi_rx_ON
                snb     transmit_ON
                setb    spi_tx_ON
                snb     spi_rx_ON
                clrb    spi_ADC_cs_pin        ;this is an example to show
                                                ;generate CS signals for a
                                                ;peripherals. This is CS fo

                snb     spi_tx_ON
                clrb    spi_DAC_cs_pin        ;CS for DAC

                sb      rising_ON              ;set the SPI_CLK polarity
                clrb    clk_state
                snb     rising_ON
                setb    clk_state
                sb      rising_ON
                clrb    spi_clk_pin
                snb     rising_ON
                setb    spi_clk_pin

                mov     w,spi_command          ;set the number of bits to
                and     w,%00001111
                mov     spi_bits_total,w
                mov     w,spi_command
                or      w,%11110000
                mov     spi_shifts_init,w
                not     spi_shifts_init
                inc     spi_shifts_init
                mov     spi_shift_counter,spi_bits_total
                clc
                rl      spi_shift_counter

                mov     spi_clk_counter,spi_clks_init ;Example for LTC159
                clc                                           ;LTC1594 input MX c
                rl      spi_clk_counter                       ;end of LTC1594 par

:set_bit_position    test    spi_shifts_init    ;initial shift before trans
                    clc      ;needed to put MSB of the w
                    sz      ;bit.7 SPI_BUFF position
                    rl      spi_buff_low
                    sz
                    rl      spi_buff
                    sz
                    djnz    spi_shifts_init,:set_bit_position

```



```

                                jmp      :master_exit      ;initialization done, first

;=====
:master_start      nop
                   test     spi_clk_counter      ;Example of the LTC1594 driv
                   jz       :master_rx           ;Example for LTC1594 ADC on
                   xor      rb,#%000000001      ;Jump to normal master rece
                   dec      spi_clk_counter      ;LTC1594 input MX control
                                           ;sclk ADC UP/DOWN

                                jmp      :master_exit      ;interrupt exit for the ADC

;=====
                                End of ltc1594 example

:master_rx      jb      clk_state,:master_tx      ;bit reception from SPI_IN_
                movb     c,spi_in_pin
                rlb      spi_buff_low
                rlb      spi_buff
                setb     spi_clk_pin
                dec      spi_shift_counter
                setb     clk_state
                jmp      :master_exit      ;bit received

:master_tx      movb     spi_out_pin,spi_buff.7      ;bit transmit
                clrb     spi_clk_pin
                dec      spi_shift_counter
                clrb     clk_state
                jmp      :master_exit      ;bit transmitted

:ms_rx_finish   sb      spi_rx_ON      ;reception completed
                jmp      :ms_tx_finish
                setb     spi_rx_complete      ;correct reception finished
                setb     spi_ADC_cs_pin      ;disable ADC CS
                clrb     spi_rx_ON
                jmp      :master_finish

:ms_tx_finish   setb     spi_DAC_cs_pin      ;disable DAC CS
                clrb     spi_tx_ON      ;transmission completed
                setb     spi_tx_complete
                jmp      :master_finish

:master_finish   clrb     spi_ms_ON
                clrb     spi_command
                clrb     spi_busy_ON

:master_exit    mov      w,spi_clk_rate
                retiw

;=====
                                END OF MASTER MODE SLOW SPI DRIVER

```